Multilabel text classification of public procurements using deep learning intent detection

ADIN SUTA
Multilabel text classification of public procurements using deep learning intent detection

ADIN SUTA
Abstract

Textual data is one of the most widespread forms of data and the amount of such data available in the world increases at a rapid rate. Text can be understood as either a sequence of characters or words, where the latter approach is the most common. With the breakthroughs within the area of applied artificial intelligence in recent years, more and more tasks are aided by automatic processing of text in various applications. The models introduced in the following sections rely on deep-learning sequence-processing in order to process and text to produce a regression algorithm for classification of what the text input refers to. We investigate and compare the performance of several model architectures along with different hyperparameters.

The data set was provided by e-Avrop, a Swedish company which hosts a web platform for posting and bidding of public procurements. It consists of titles and descriptions of Swedish public procurements posted on the website of e-Avrop, along with the respective category/categories of each text.

When the texts are described by several categories (multi label case) we suggest a deep learning sequence-processing regression algorithm, where a set of deep learning classifiers are used. Each model uses one of the several labels in the multi label case, along with the text input to produce a set of text - label observation pairs. The goal becomes to investigate whether these classifiers can carry out different levels of intent, an intent which should theoretically be imposed by the different training data sets used by each of the individual deep learning classifiers.
Referat

Data i form av text är en av de mest utbredda formerna av data och mängden tillgänglig textdata runt om i världen ökar i snabb takt. Text kan tolkas som en följd av bokstäver eller ord, där tolkning av text i form av ordföljder är absolut vanligast. Genombrott inom artificiell intelligens under de senaste åren har medfört att fler och fler arbetsuppgifter med koppling till text assisteras av automatisk textbearbetning. Modellerna som introduceras i denna uppsats är baserade på djupa artificiella neuronät med sekventiell bearbetning av textdata, som med hjälp av regression förutspår tillhörande ämnesområde för den inmatade texten. Flera modeller och tillhörande hyperparametrar utreds och jämförs enligt prestanda.

Datamängden som använts är tillhandahållet av e-Avrop, ett svenskt företag som erbjuder en webbtjänst för offentliggörande och budgivning av offentliga upphandlingar. Datamängden består av titlar, beskrivningar samt tillhörande ämneskategorier för offentliga upphandlingar inom Sverige, tagna från e-Avrops webbtjänst.

När texterna är märkta med ett flertal kategorier, föreslås en algoritm baserad på ett djupt artificiellt neuronat med sekventiell bearbetning, där en mängd klassificeringsmodeller används. Varje sådan modell använder en av de märkta kategorierna tillsammans med den tillhörande texten, som skapar en mängd av text-kategori par. Målet är att utreda huruvida dessa klassificerare kan upphöja olika former av uppsätt som teoretiskt sett borde vara medfört från de olika datamängderna modellerna mottagit.
Acknowledgements

I would like to thank my family for supporting me in the period of working on this thesis, and during the entirety of my studies. I would like to thank Joakim Poromaa Helger for providing the data set and for being available to explain all my questions regarding it. Finally I would like to thank Tatjana Pavlenko for the help, feedback and guidance regarding the work on this thesis.
## Contents

1 Introduction
   
   1.1 Background ........................................ 2
   
   1.1.1 Company description and background information  2
   
   1.1.2 Description of the CPV code system ............ 2
   
   1.2 Problem ......................................... 3
   
   1.3 Purpose ......................................... 3
   
   1.4 Goal ............................................ 4
   
   1.5 Methodology .................................... 5
   
   1.6 Delimitation ................................... 5
   
   1.7 Outline ....................................... 6

2 Extended background
   
   2.1 Natural language processing ......................... 7
   
   2.2 Word embedding ................................... 7
   
   2.2.1 Word embedding from n-grams .................... 8
   
   2.3 Artificial Neural network .......................... 10
   
   2.3.1 Fully connected neural network .................. 10
   
   2.3.2 Recurrent neural network ....................... 12
   
   2.3.3 Attention in neural networks .................... 17
   
   2.3.4 Training of artificial neural networks .......... 18
   
   2.3.5 Neural networks for regression .................. 20

3 Methods
   
   3.1 Data ........................................... 22
   
   3.2 Input space ...................................... 24
   
   3.3 Output space ..................................... 26
   
   3.4 Loss function .................................... 27
   
   3.5 Validation ...................................... 27
   
   3.6 Training models .................................. 28
   
   3.6.1 Single label case ................................ 28
   
   3.6.2 Multi label case ................................ 29

4 Results .............................................. 32
4.1 Training of deep learning models in the single labels case .................. 32
4.2 Performance of the deep learning models in using the multi label data set .................................................................................. 35

5 Discussion .................................................................................. 42
5.1 Performance of neural networks ................................................. 42
  5.1.1 Input space ........................................................................... 42
  5.1.2 Output space ......................................................................... 43
5.2 Deep learning classifiers with intent ........................................... 43
5.3 Future work .............................................................................. 44

6 Conclusion ............................................................................... 45

Bibliography ................................................................................. 46
List of Figures

1.1 A part of the CPV-code tree from the e-Avrop website search function. 3

2.1 An illustration of the word distance in the vector space. Reproduced from *Vector Representations of Words* [10]. 8

2.2 An single neuron with four inputs. Reproduced from Goldberg and Hirst [15]. 11

2.3 Feed-forward neural network with two hidden layers. Reproduced from Goldberg and Hirst [15]. 11

2.4 Graphical representation of an RNN (recursive). Reproduced from Goldberg and Hirst [15]. 13

2.5 Graphical representation of the computing of a biRNN of the word *jumped*. The input sequence to the biRNN is the sentence “the brown fox jumped over the dog. Reproduced from Goldberg and Hirst [15]. 14

2.6 Computing the biRNN* for the sentence “the brown fox jumped.” Reproduced from Goldberg and Hirst [15]. 15

3.1 Frequency of observations in each of the most general categories. All observations of the 30 000 public procurements are plotted. 23

3.2 Frequency of observations in each of the most general categories. All observations of the 30 000 public procurements are plotted. 24

3.3 Histogram made out of all observation of number of words per title. 25

3.4 Histogram made out of all observation of number of words per description. 25

3.5 Histogram of number of codes per CPV code description. 26

3.6 Cross Validation - example of data partition if 20 data points into 4 subsets, train on 3 subsets and test on 1 subset. Reproduced from *K-fold and other cross-validation techniques* [31]. 28

3.7 Depth of nodes illustrated. 29

4.1 Training and validation loss of the GRU\_max model. 32

4.2 Training and validation loss of the LSTM\_max model. 33

4.3 Training and validation loss of the LSTM\_conv model. 33

4.4 Training and validation loss of the GRU\_att model. 34

4.5 Training and validation loss of the LSTM\_att model. 34
4.6 Training and validation loss of the GRU\_att model for the "general" data set. 35
4.7 Training and validation loss of the GRU\_att model for the "Ties\_1" data set. 36
4.8 Training and validation loss of the GRU\_att model for the "Ties\_2" data set. 36
4.9 Training and validation loss of the GRU\_att model for the "detailed" data set. 37
4.10 Training and validation loss of the GRU\_att model for the "general" data set. 38
4.11 Training and validation loss of the GRU\_att model for the "Ties\_1" data set. 38
4.12 Training and validation loss of the GRU\_att model for the "Ties\_2" data set. 39
4.13 Training and validation loss of the GRU\_att model for the "detailed" data set. 39
4.14 Histogram of classification depth for the general/detailed model. 40
List of Tables

3.1 Table of models tested in the single-label case. .......................... 28
3.2 Table of input/output dimensions of the ANN. ............................ 29
4.1 Table of difference in average classification depth between the different deep learning classifiers. ............................. 41
Chapter 1

Introduction

The history of writing is the development of expressing language by letters or other marks which traces back to at least two ancient civilizations [1]. Both in ancient Sumer (Mesopotamia) between 3400 and 3300 BC and much later in Mesoamerica at around 300 BC, the concept of writing was conceived independently [2]. Written language enables humans to capture and convey complex messages which in turn allows for spreading of information and knowledge around the world. Today, and since the beginning of the first written text, we are accumulating more and more information over time which with the advancement of information technology allows for storage of unprecedented amounts of data, while becoming more accessible than ever before.

To program computers to understand human written language is a task which has been in development ever since the initial rise of computational machines. Natural language processing (NLP) is the subfield of computer science, information engineering and artificial intelligence concerned with the interactions between human (natural) languages and computers, in particular processing and analyzing large amounts natural language data. In many industries, effective automated text classification could be of great use to quickly sort through and analyze large amounts of data, assisting humans in such tasks or perhaps some day completely replacing text classification tasks which rely on human analysis.

Recently, state of the art text classification algorithms has often relied on numerical vector representations of natural language words (word embeddings) [3]. These models are able to represent words from natural language as numbers, allowing this information to be processed by a computer via a classification algorithm. The word representations are trained using large corpuses of millions of words, where a vector space is created and word similarity is easily captured. This is useful in deep learning, where words are transformed into numbers and fed into an artificial neural network (ANN).

In this thesis, the task of automating manual text classification was implemented to handle the problem of categorization of public procurements in Sweden. Using the CPV code (Common public Procurement Vocabulary code) categorization system,
which is standardized in the EU, titles and descriptions of public procurements were used in order to classify them into one or several categories.

1.1 Background

1.1.1 Company description and background information

E-avrops a Swedish company that provides a web platform for public procurements. The system enables a fully digitized procurement and purchasing process and is used by a large number of Swedish municipalities and government agencies, including Swedavia, the Swedish Work Environment Authority, PRV and others. The company has 13 employees in Sweden with its head office in Danderyd.

1.1.2 Description of the CPV code system

In order to make each public procurement easily accessible and understandable for clients in EU, the Common Procurement Vocabulary (CPV) was introduced [4]. It is mandatory to classify each public procurement with regards to the CPV system since the 1st of February 2006 in EU.

The idea of the classification is that the contracting authority must choose the CPV code that best matches the current need and the object of the procurement. It is the object of the procurement that governs which CPV code to specify. When appropriate and for better coding, multiple CPV codes can be used. If it is not possible to find a suitable and exact code, the contracting authority may instead make a reference to a suitable main group, subgroup or category.

In order for potential suppliers to be able to find relevant procurements, it is important that the contracting authority makes a correct CPV coding of the procurement, and that the supplier monitors the correct CPV codes based on what one wishes to deliver. Although the purchaser has the task of choosing the correct CPV code, this coding can be defective. The potential suppliers can often lack knowledge of the CPV classification system, which also means they miss out on contracts.

Each language has its own definition list, where the CPV code and its description exist. Such a description for the CPV code 30100000 may look like the following:

1. 30100000 - Kontorsmaskiner, kontorsutrustning och kontorsmateriel, utom datorer, skrivare och möbler (swedish description)

2. 30100000 - Office machinery, equipment and supplies except computers, printers and furniture (english description)

A Swedish contractor can thus advertise with the code 30100000 and ensure that even an English supplier will find it, provided that the English supplier monitors the specific code.

Since the codes are hierarchically structured, the collection is usually called a CPV tree. The following example shows how the structure can look.
1.2. PROBLEM

1. **30000000** - Office machinery, equipment and supplies except computers, printers and furniture

1.1. **30100000** Office machines, office equipment and office supplies, except computers, printers and furniture

1.2. **30200000** Computers and computer supplies

Figure 1.1: A part of the CPV-code tree from the e-Avrop website search function.

1.2 Problem

When classification of public procurements is done by a human, the classification process is straightforward: the number of CPV codes used to classify a public procurement should be enough so that the number of codes correctly capture all the categories in which the procurement belongs while excluding categories which are superfluous. This is therefore a case of multi label classification where each observation (title and description of the public procurement) belongs to one or several labels. Naturally, a classification needs to not only have correct labels but also correct amount of labels, which the classification algorithm needs to account for.

1.3 Purpose

The purpose of this thesis is to propose a way of multi label text classification with deep learning and pre-trained word embeddings. By choosing different training sets
for several ANN classifiers, the ANN’s focus on different parts of the data and have different intent. The main research question is therefore:

**Can we construct a set of deep learning text classifiers where each classifier captures different characteristics of a text and provides different behaviour of classification?**

The above research question will be evaluated according to 1) effectiveness, 2) applicability (on the specific data set used in the thesis).

1. **Effectiveness:** Performance of the models and how well the models capture different characteristics of a text.

2. **Applicability:** Applicability means the right amount of classifiers, observations and labels to be used.

Evaluating different deep learning models according to performance on the particular data set of public procurements and their respective differences is a way of measuring effectiveness. Note that applicability is subject to interpretation, but motivation of why certain methods and models are used is a way of presenting the applicability on a particular multi label data set. Different multi label text classification problems require a different amount of classifiers in order to capture different levels of intent.

### 1.4 Goal

The goal of this thesis is related to the purpose discussed in the previous section. In order to be able to answer the research question and evaluate the answer the following goals need to be achieved:

1. Analysis of the particular data set at hand, motivating the right amount classifiers and the input/output of the deep learning model.

2. Suitable choice of structure for the deep learning text classification model in order to provide a base for the multi label case.

3. Analysis and evaluation of performance and applicability to be able to motivate the answer to the research question.

In this work, a way of handling multi label text classification is presented which on one hand relies on the deep learning model, and on the other hand relies on the feature extractions and understanding of the particular problem at hand. The former is mainly evaluated using the comparison of different models and is required to evaluate the latter. In order to be able to handle the multi label case of text classification, the deep learning model must be chosen with care, with the applicability being a challenge since the measurement of applicability is not easily defined.
1.5 Methodology

Today, there are several high-quality pre-trained word embeddings available that could be used as a transformation from natural language into vector representations of words. In this thesis, fastText from facebook was chosen as the input word embedding for the ANN used to classify public procurements. The reason is that the skip-gram model on which fastText is based on is state of the art while at the same time being trained at n-gram level. When the word embeddings are trained, they are not only trained on words but on word decompositions, so called n-grams. This is particularly useful in languages such as Swedish and German where compounding of words is very common. For this reason, out-of-vocabulary words are represented by their n-grams and always have a vector representation while misspellings are also evaluated by their n-gram. This makes fastText a robust word embedding, especially useful as a tool to interpret the Swedish language.

The idea behind the architecture of the ANN models is that natural language is sequential in its nature (the next word depend on the previous word) which makes sequential encoders such as bidirectional LSTM and GRU layers viable in the first stage of the encoding process. The summation (convolutions, attention or capsule layers for example) are used as a way of summarizing the encoded text, while at the end of the models classifying with a fully connected layer. These models have become standard in the NLP community in the past few years when using neural networks for text classification tasks.

Based on these results, an ANN model is presented which performs best against other models of similar structure, by dividing the data into training and validation sets. In order to choose a proper amount of classifiers used in the final multi label classifier, the data is carefully analyzed to evaluate the applicability of these classifiers on the public procurements. The data is analyzed according to how many labels per observations is most common, the structure and behaviour these labels usually follow.

1.6 Delimitation

In this thesis we only experiment with fastText word embeddings while other word embeddings are not used. Performance of the models may be different than if other word embeddings would be used.

The ANN models are similar in nature, because of the assumption that sequential data like text works best with a sequential encoder, a summation part and a final classification part. It is possible that other structures are more effective, but a delimitation was made in how many ANN models are tested and compared.
1.7 Outline

The structure of the thesis is as follows. In chapter 2, the mathematical theory of word embeddings and deep learning methods are presented. In chapter 3, the data is visualized in a way that the reader can understand the motivation behind the chosen methods, motivating the choice of models used later on. We then explain the chosen methods of handling input data, output data and single/multi label data. In chapter 4, the results are presented. The results are divided into two categories, the single label and the multi label case. In chapter 5, these results are discussed and evaluated according to the research question. Chapter 6 contains conclusions from the experiments along with proposed valid future works.
Chapter 2

Extended background

2.1 Natural language processing

Natural language processing (NLP) also called computational linguistics is the subfield of computer science concerned with the interactions between computer language and human (natural language). The earliest research in NLP is generally considered to have started in the 1950s where the so called Turing test was introduced, proposing a test in machine intelligence which to this day is widely known. The Turing test, developed by Alan Turing in 1950, is a test of a machine’s ability to possess indistinguishable human behaviour and to appear to have human-like intelligence and interaction. The test is based on a machine’s ability to analyze, process and contribute in an interaction based in a human language [5].

Up until the 1980s, the area of NLP relied on sets of complex hand written rules in order to execute decision making. In the late 1980s however, probabilistic models where introduced with machine learning algorithms, often based on decision trees or hidden Markov models [6]. These models required a higher amount of computational power which was had not been available until then. Today, computers have orders of magnitudes stronger computing power with more and more natural language data being accumulated over time, especially with the global usage of internet in recent years.

As the area of NLP is a crossover between natural language and computer language, an obvious question becomes: How do we transform natural language into a language that a computer can understand and process? In the following section, this transformation is thoroughly explained.

2.2 Word embedding

A word embedding is a dense numerical vector representation of a word, where features of words or phrases are mapped to vectors of real numbers. The majority of work advocate the use of dense, trainable embedding vectors for all features [7][8][9]. A model that transforms an input word to a fixed-length dense vector
is called a word embedding model. Today, word embedding models use various training methods, mostly neural networks, dimensionality reduction, probabilistic models or word co-occurrence matrices. Large data sets are used in order to capture complex linguistic semantics.

Figure 2.1: An illustration of the word distance in the vector space. Reproduced from Vector Representations of Words [10].

Semantically, similar words tend to have similar embeddings and the similarity is measured using cosine similarity. In figure 2.1 this is illustrated by some famous examples. Here, male-female relationship between words are captured where for example the word embeddings of: "king" − "man" + "woman" = "queen". Other semantics like verb tense or relationships like "country" − "capital" and many more linguistic linkages are captured in the vector space as illustrated in figure 2.1. These linguistic semantics are captured in the vector space, where similar words tend to have similar vector representations.

2.2.1 Word embedding from n-grams

General skip-gram model

The general continuous skip-gram model for obtaining word embeddings is introduced by Mikolov et al. [11]. Given a word vocabulary of size $W$, where a word is identified by its index $w \in \{1, \ldots, W\}$ the goal is to learn a vectorial representation of each word $w$. Word representations are trained to predict well words that appear in its context. More formally, given a large training corpus represented as a sequence of words $w_1, \ldots, w_T$, the objective of the skipgram model is to maximize the following log likelihood:

$$
\sum_{t=1}^{T} \sum_{c \in C_t} \log p(w_c|w_t),
$$

(2.1)

where the context $C_t$ is the set of indices of words surrounding word $w_t$. The probability of observing a context word $w_c$ given $w_t$ will be parameterized using the
2.2. WORD EMBEDDING

aforementioned word vectors. For now, let us consider that we are given a scoring function \( \rho \) which maps pairs of (word, context) to scores in \( \mathbb{R} \). One possible choice to define the probability of a context word is the softmax:

\[
p(w_c|w_t) = \frac{e^{\rho(w_t, w_c)}}{\sum_{j=1}^{W} e^{\rho(w_t, j)}}.
\] (2.2)

The problem of predicting context words can instead be framed as a set of independent binary classification tasks. The goal then becomes to predict the presence or absence of context words. For the word at position \( t \) we consider all context words as positive examples and sample negatives at random from the dictionary. For a chosen context position \( c \), using the binary logistic loss, we obtain the following negative log-likelihood:

\[
\log \left(1 + e^{-\rho(w_t, w_c)}\right) + \sum_{n \in \mathcal{N}_{t,c}} \log \left(1 + e^{\rho(w_t, n)}\right),
\] (2.3)

where \( \mathcal{N}_{t,c} \) is a set of negative examples sampled from the vocabulary. By denoting the logistic loss function \( \ell : x \mapsto \log (1 + e^{-x}) \), we can re-write the objective as:

\[
\sum_{t=1}^{T} \left[ \sum_{c \in \mathcal{C}_t} \ell (\rho (w_t, w_c)) + \sum_{n \in \mathcal{N}_{t,c}} \ell (-\rho (w_t, n)) \right].
\] (2.4)

A natural parameterization for the scoring function \( \rho \) between a word \( w_t \) and a context word \( w_c \) is to use word vectors. Let us define for each word \( w \) in the vocabulary two vectors \( u_w \) and \( v_w \) in \( \mathbb{R}^d \). These two vectors are sometimes referred to as input and output in the literature. In particular, we have vectors \( u_{w_t} \) and \( v_{w_c} \), corresponding, respectively, to words \( w_t \) and \( w_c \). Then the score can be computed as the scalar product between word and context vectors as \( \rho (w_t, w_c) = u_{w_t}^T v_{w_c} \).

The model described in this section is the skipgram model with negative sampling, introduced by Mikolov et al. [11].

Subword model

By using a distinct vector representation of each word, the skipgram model ignores the internal structure of words. Therefore, languages that use compound words (words that are put together using shorter words) have issues modelling the semantic complexity of the language. In this section, a different scoring function \( \rho_{ngram} \) is proposed, in order to take into account sub word information.

Each word \( w \) is represented as a bag of character \( n \)-grams. We add special boundary symbols \(< \) and \( > \) at the beginning and end of words, allowing to distinguish prefixed and suffixes from other character sequences. We also include the word \( w \) itself in the set of its \( n \)-grams, to learn the representation for each word (in addition to character \( n \)-grams). Taking the word \( where \) and \( n = 3 \) as an example, it will be represented by the character \( n \)-grams:
<\textit{wh}, \textit{whe}, \textit{her}, \textit{ere}, \textit{re}> \\
and the special sequence \\
<\textit{where}>.

Note that the sequence <\textit{her}>, corresponding to the word \textit{her} is different from the tri-gram \textit{her} from the word \textit{where}. Different sets of \(n\)-grams could be considered, for examples taking all prefixes and suffixes.

Suppose that you are given a dictionary of \(n\)-grams of size \(G\). Given a word \(w\), let us denote by \(G_w \subset \{1, \ldots, G\}\) the set of \(n\)-grams appearing in \(w\). We associate a vector representation \(\tau_g\) to each \(n\)-gram \(g\). We represent a word by the sum of the vector representations of its \(n\)-grams. We thus obtain the scoring function:

\[
\rho_{\text{ngram}}(w, c) = \sum_{g \in G_w} \tau_g^T v_c.
\] (2.5)

This simple model allows for sharing the representations across words, thus allowing to learn reliable representations for rare words [12].

In this thesis, the word embedding used is fastText, developed by facebook. Trained on the entire Swedish wikipedia corpus (available for 157 languages), it utilizes the subword model which makes it robust for out-of-vocabulary words such as compound words, or misspellings [13].

### 2.3 Artificial Neural network

Artificial neural networks (ANN) are mathematical computing units, vaguely inspired by the neurons of brains in humans and animals. While the connections between artificial neural networks and the brain are in fact rather slim, we repeat the metaphor in this section for completeness. An ANN is based on a collection of connected nodes or units called artificial neurons.

Typically, artificial neurons are divided into layers where each layer has a different sub problem to solve in order to solve the problem as a whole. Below, we go through the layers that are used in the many NLP classification tasks [14], as well as in this thesis.

#### 2.3.1 Fully connected neural network

In the metaphor, a neuron is a computational unit that has scalar inputs and outputs. Each input has an associated weight. The neuron multiplies each input by its weight, and then sums\(^1\) them, applies a nonlinear function to the result and passes it to its output. The neurons are connected to each other, forming a network; the

\(^1\)While summing is the most common operation, other functions, such as a max, are also possible.
2.3. ARTIFICIAL NEURAL NETWORK

output of a neuron may feed into the outputs of one or more neurons. Such networks were shown to be very capable computing devices.

![Figure 2.2: An single neuron with four inputs. Reproduced from Goldberg and Hirst [15].](image)

A typical feed-forward neural network may be drawn as in figure 2.3 where each neuron is connected to all of the neurons in the next layer - this is called a *fully connected layer* or an *affine layer*. Each circle is a neuron with incoming arrows being the neuron’s inputs and outgoing arrows being the neuron’s outputs. Each arrow carries a weight, reflecting its importance (not shown).

![Figure 2.3: Feed-forward neural network with two hidden layers. Reproduced from Goldberg and Hirst [15].](image)
The simplest neural network is called a perceptron. It is simply a linear model:

$$\text{NN}_{\text{Perceptron}}(x) = x^\top W + b^\top$$

where $W$ is the weight matrix and $b$ is a bias term. In order to go beyond linear functions, we introduce a nonlinear hidden layer (the network in figure 2.3 has two such layers), resulting in the Multi Layer Perceptron with one hidden-layer (MLP1).

A feed forward neural network with one hidden-layer has the form:

$$\text{NN}_{\text{MLP1}}(x) = g(x^\top W_1 + b_1^\top) W_2 + b_2^\top$$

Here $W_1$ and $b_1$ are a matrix and bias term for the first linear transformation of the input, $g$ is a nonlinear function that is applied element-wise (also called a nonlinearity or an activation function), and $W_2$ and $b_2$ are the matrix and bias term for a second linear transform. The nonlinear activation function $g$ has a crucial role in the network’s ability to represent complex functions.

Breaking it down, $x^\top W_1 + b_1^\top$ is a linear transformation of the input $x$ from $d_{in}$ dimensions to $d_1$ dimensions, and the matrix $W_2$ together with a bias vector $b_2$ are then used to transform the result into the $d_2$ dimensional output vector. The same principle follows no matter the layer count, each output of a layer is the input to the next where only the last layer outputs the result [15].

### 2.3.2 Recurrent neural network

Recurrent neural networks (RNN) are constructed for dealing with sequential modeling, such as sound, voice, text etc [16]. It is convenient to view recurrent neural networks from the point of view of hidden Markov models.

We use $x_{i:j}$ to denote the sequence of vectors $x_i, \ldots, x_j$. On a high-level, the RNN is a function that takes as input an arbitrary length ordered sequence of $n$ $d_{in}$-dimensional vectors at time $t$, $x_t = x_{1:n} = x_1, x_2, \ldots, x_n$, $(x_i \in \mathbb{R}^{d_{in}})$ and returns as output a single $d_{out}$ dimensional vector $y_n \in \mathbb{R}^{d_{out}}$:

$$y_n = \text{RNN}(x_{1:n})$$

$x_i \in \mathbb{R}^{d_{in}}$ $y_n \in \mathbb{R}^{d_{out}}$.  

(2.8)

This implicitly defines an output vector $y_i$ for each prefix $x_{1:i}$ of the sequence $x_{1:n}$. We denote by $\text{RNN}^*$ the function returning this sequence:

$$\text{RNN}^*(x_{1:n}; s_0) = y_{1:n}$$

$$y_i = O(s_i)$$

$$s_i = R(s_{i-1}, x_i)$$

(2.9)
2.3. ARTIFICIAL NEURAL NETWORK

\[ x_i \in \mathbb{R}^{d_{in}}, \quad y_i \in \mathbb{R}^{d_{out}}, \quad s_i \in \mathbb{R}^{f(d_{out})}. \]

The output vector \( y_n \) is then used for further prediction. For example, a model for predicting the conditional probability of an event \( e \) given the sequence \( x_{1:n} \) can be defined as \( p(e = j | x_{1:n}) = \text{softmax} \left( \text{RNN} \left( x_{1:n} \right) \cdot W + b \right)_{(j)}, \) the \( j \)th element in the output vector resulting from the softmax operation over a linear transformation of the RNN encoding \( y_n = \text{RNN} \left( x_{1:n} \right) \). The RNN function provides a framework for conditioning on the entire history of \( x_1, x_2, \ldots, x_i \) without resorting to the Markov assumption which was traditionally used for modeling sequences. Looking in a bit more detail, the RNN is defined recursively, by means of a function \( R \) taking as input a state vector \( s_{i-1} \) and an input vector \( x_i \) and returning a new state vector \( s_i \). The state vector \( s_i \) is then mapped to an output vector \( y_i \) using a simple deterministic function \( O(\cdot) \). The base of the recursion is an initial state vector \( s_0 \), which is also an input to the RNN. For brevity, we often omit the initial vector \( s_0 \), or assume it is the zero vector. When constructing an RNN, much like when constructing a feed forward network, one has to specify the dimension of the inputs \( x_i \) as well as the dimensions of the outputs \( y_i \). The dimensions of the states \( s_i \) are a function of the output dimension.\(^2\)

The functions \( R \) and \( O \) are the same across the sequence positions, but the RNN keeps track of the states of computation through the state vector \( s_i \) that is kept and being passed across invocations of \( R \). Graphically, the RNN has been traditionally presented as in figure 2.4 [15].

![Graphical representation of an RNN (recursive). Reproduced from Goldberg and Hirst [15].](image)

**Bidirectional RNN**

A useful elaboration of an RNN is a bidirectional-RNN (also commonly referred to as biRNN) [17][18]. Consider the task of sequence tagging over a sentence \( x_1, \ldots, x_n \). An RNN allows us to compute a function of the \( i \)th word \( x_i \) based on the past which

\(^2\)While RNN architectures in which the state dimension is independent of the output dimension are possible, the current popular architectures, including the Simple RNN, the LSTM, and the GRU do not follow this flexibility.
are the words $x_{1:i}$ up to and including it. However, the following words $x_{i+1:n}$ may also be useful for prediction, as is evident by the common sliding-window approach in which the context of the focus word is categorized based on a window of $k$ words surrounding it. Much like the RNN relaxes the Markov assumption and allows looking arbitrarily back into the past, the biRNN relaxes the fixed window size assumption, allowing to look arbitrarily far at both the past and the future within the sequence.

Consider an input sequence $x_{1:n}$. The biRNN works by maintaining two separate states $s^f_i$ and $s^b_i$ for each input position $i$. The forward state $s^f_i$ is based on $x_1, x_2, \ldots, x_i$, while the second RNN $(R^b, O^b)$ is fed the input sequence in reverse. The state representation $s_i$ then composed of both the forward and backward states. The output at position $i$ is based on the concatenation of the two output vectors $y_i = [y_i^f; y_i^b] = [O^f(s^f_i); O^b(s^b_i)]$, taking into account both the past and the future. In other words, $y_i$, the biRNN encoding of the $i$th word in a sequence is the concatenation of two RNNs, one reading the sequence from the beginning, and the other reading it from the end.

We define $\text{biRNN}(x_{1:n}, i)$ to be the output vector corresponding to the $i$th sequence position:

$$\text{biRNN}(x_{1:n}, i) = y_i = \left[ \text{RNN}^f(x_{1:i}); \text{RNN}^b(x_{n:i}) \right].$$

(2.10)

The vector $y_i$ can then be used directly for prediction, or fed as part of the input to a more complex network. While the two RNNs are run independently of each other, the error gradients at position $i$ will flow both forward and backward through the two RNNs. Feeding the vector $y_i$ through an MLP prior to prediction will further mix the forward and backward signals. Visual representation of the biRNN architecture is given in Figure 2.5.

Figure 2.5: Graphical representation of the computing of a biRNN of the word jumped. The input sequence to the biRNN is the sentence “the brown fox jumped over the dog. Reproduced from Goldberg and Hirst [15].
Similarly to the RNN case, we also define biRNN* \( (x_{1:n}) \) as the sequence of vectors \( y_{1:n} \):

\[
\text{biRNN}^* \left( x_{1:n} \right) = y_{1:n} = \text{biRNN} \left( x_{1:n}, 1 \right), \ldots, \text{biRNN} \left( x_{1:n}, n \right).
\] (2.11)

The \( n \) output vectors \( y_{i:n} \) can be efficiently computed in linear time by first running the forward and backward RNNs, and then concatenating the relevant outputs. This architecture is depicted in Figure 2.6.

The biRNN is very effective for tagging tasks, in which each input vector corresponds to one output vector. It is also useful as a general-purpose trainable feature-extracting component, that can be used whenever a window around a given word is required [15].

**Problems in training the RNN**

The simplest RNN formulation that is sensitive to the ordering of elements in the sequence is known as an Elman Network or Simple-RNN (S-RNN). The S-RNN was proposed by Elman [16] and explored for use in language modeling by Ueúí et al. [19]. The S-RNN takes the following form:

\[
s_i = R_{\text{SRNN}} \left( x_i, s_{i-1} \right) = g \left( s_{i-1}^T W_s + x_i^T W_x + b^T \right)
\]

\[
y_i = O_{\text{SRNN}} \left( s_i \right) = s_i
\]

(2.12)

\[s_i, y_i \in \mathbb{R}^{d_s}, \ x_i \in \mathbb{R}^{d_x}, \ W_x \in \mathbb{R}^{d_x \times d_s}, \ W_s \in \mathbb{R}^{d_s \times d_s}, \ b \in \mathbb{R}^{d_s}.
\]

That is, the state \( s_{i-1} \) and the input \( x_i \) are each linearly transformed with corresponding weight matrices, the results are added (together with a bias term) and
then passed through a nonlinear activation function $g$ (commonly tanh or ReLU). The S-RNN is hard to train effectively because of the vanishing gradient problem (Bengio, Simard, and Frasconi [20]). Error signals (gradients) in later steps in the sequence diminish quickly in the backpropagation process, and do not reach earlier input signals, making it hard for the S-RNN to capture long-range dependencies. Gating based architectures such as the Long-Short-Term-Memory (LSTM) or Gated-Recurent-Units (GRU) are designed to solve this deficiency [15].

**LSTM**

The LSTM architecture was designed to solve the vanishing gradients problem and is the first to introduce the gating mechanism [21]. The LSTM architecture explicitly splits the state vector $s_t$ into two halves, where one half is treated as "memory cells" and the other is working memory. The memory cells are designed to preserve the memory and also the error gradients across time and are controlled through differentiable gating components - smooth mathematical functions that simulate logical gates. At each input state, a gate is used to decide how much of the new input should be written to the memory cell, and how much of the current content of the memory cell should be forgotten. Mathematically, the LSTM architecture is defined as:

$$s_j = R_{LSTM}(s_{j-1}, x_j) = [q_j; h_j]$$

$$q_j = f \odot q_{j-1} + i \odot z$$

$$h_j = o \odot \tanh(q_j)$$

$$i = \sigma(x_j^\top W_{xi} + h_{j-1}^\top W_{hi})$$

$$f = \sigma(x_j^\top W_{xf} + h_{j-1}^\top W_{hf})$$

$$o = \sigma(x_j^\top W_{xo} + h_{j-1}^\top W_{ho})$$

$$z = \tanh(x_j^\top W_{xz} + h_{j-1}^\top W_{hz})$$

Here $\odot$ is the element wise multiplication operation and $\sigma$ is the sigmoid activation function. The state at time $j$ is composed of two vectors, $q_j$ and $h_j$ where $q_j$ is the memory component and $h_j$ is the hidden state component. There are three gates, $i$, $f$ and $o$, controlling for input, forget and output. The gate values are computed based on linear combinations of the current input $x_j$ and the previous hidden state $h_{j-1}$, passed through a tanh activation function. The gating mechanisms allow for gradients related to the memory part $q_j$ to stay high across very long time ranges [15].
2.3. ARTIFICIAL NEURAL NETWORK

GRU

The LSTM architecture is currently the most successful type of RNN, but it is also quite complicated. A simpler and less computationally expensive RNN architecture is the Gated Recurrent Unit (GRU), introduced by Cho et al. [22]. Like the LSTM, the GRU is based on a gating mechanism, but with fewer gates and without a separate memory component. The mathematical definition of the GRU is defined as:

\[
\begin{align*}
    s_j &= R_{GRU}(s_{j-1}, x_j) = (1 - z) \odot s_{j-1} + z \odot \tilde{s}_j \\
    z &= \sigma(x_j^\top W_{xz} + s_{j-1}^\top W_{sz}) \\
    r &= \sigma(x_j^\top W_{xr} + s_{j-1}^\top W_{sr}) \\
    \tilde{s}_j &= \tanh(x_j^\top W_{xs} + (r \odot s_{j-1})^\top W_{sg}) \\
    y_j &= O_{GRU}(s_j) = s_j
\end{align*}
\]

(2.14)

One gate (r) is used to control access to the previous state \(s_{j-1}\) and compute the proposed update \(\tilde{s}_j\). The updated state \(s_j\) (which also serves as the output \(y_j\)) is then determined based on an interpolation of the previous state \(s_{j-1}\) and the proposal \(\tilde{s}_j\), where the proportions of the interpolation are controlled using the gate \(z\). The GRU was shown to be effective in language modeling and machine translation. However, the jury is still out between the GRU, the LSTM and possible alternative RNN architectures, and the subject is actively researched [15].

2.3.3 Attention in neural networks

In NLP, encoder-decoder networks are commonly used in text classification. A function is first used to encode the input (map the raw input to features) which sends the information to a decoder. The task of the decoder is to use the encoded information to produce something that can be used as a classification.

Using an RNN-architecture as an encoder, the vector that the RNN produces is forced to contain all the information required to be used for classification. This can in many cases be improved by adding an attention mechanism [23]. The work of Luong, Pham, and Manning [24] explores some of them in the context of machine translation. The attention mechanism is used as a decoder in order to decide on which parts of the encoding input it should focus. More concretely, the encoder-decoder with attention architecture encodes a length \(n\) input sequence \(x_{1:n}\) using a biRNN, producing \(n\) vectors \(\varphi_{1:n}\):

\[
\varphi_{1:n} = ENC(x_{1:n}) = biRNN^*(x_{1:n}).
\]

At any stage \(j\) the decoder chooses which of the vectors \(\varphi_{1:n}\) it should attend to, resulting in a focused context vector \(\varphi^j = \text{attend}(\varphi_{1:n}, f_{1:j})\) us then used for
CHAPTER 2. EXTENDED BACKGROUND

conditioning the generation at step $j$:

\[
\begin{align*}
  p(t_{j+1} = k | \hat{t}_1: j, x_{1:n} = f(O(s_{j+1}))) \\
  s_{j+1} = R(s_j, [\hat{t}_j; \varphi^j]) \\
  \varphi^j = \text{attend}(\varphi_{1:n}, \hat{t}_1: j) \\
  \hat{t}_j \sim p(t_j | \hat{t}_{1:j-1}, x_{1:n}).
\end{align*}
\]

(2.15)

The attention mechanism is thoroughly described in Bahdanau, Cho, and Bengio [23], who were first to introduce attention in the context of sequence to sequence generation. The function $\text{attend}(\cdot, \cdot)$ is a trainable, parametrized function. The implemented attention mechanism is soft, meaning that at each stage the decoder sees a weighted average of the vectors $\varphi_{1:n}$, where the weights are chosen by the attention mechanism. More formally, at stage $j$ the soft attention produces a mixture vector $\varphi^j$:

\[
\varphi^j = \sum_{i=1}^{n} \alpha^j_i \cdot \varphi_i.
\]

$\alpha^j \in \mathbb{R}^n$ is the vector of attention weights for stage $j$, whose elements $\alpha^j_i$ are all positive and sum to one.

The values $\alpha^j_i$ are produced in a two stage process: first, unnormalized attention weights $\alpha^j_i$ are produced using a feed-forward network $\text{MLP}^{\text{att}}$ taking account the decoder state at time $j$ and each of the vectors $\varphi^j$:

\[
\begin{align*}
  \alpha^j & = \alpha^j_1, \ldots, \alpha^j_n = \\
  & = \text{MLP}^{\text{att}}([s_j; \varphi_1]), \ldots, \text{MLP}^{\text{att}}([s_j; \varphi_n]).
\end{align*}
\]

(2.16)

The unnormalized weights $\alpha^j$ are then normalized into a probability distribution using the soft-max function:

\[
\alpha^j = \text{softmax}(\alpha^j_1, \ldots, \alpha^j_n).
\]

The biRNN is used as an encoder to translate a sentence $x_{1:n}$ into the context vectors $\varphi_{1:n}$ because of the sequential context property of the biRNN. The biRNN produces a window focused around the input item $x_i$ and not the item itself, which provides contextual information from the source input $x_i$. When the attention mechanism is jointly trained with the biRNN encoder, the biRNN encoder may learn to encode the position of $x_i$ within the sequence, and the decoder could use this information to access the elements in order, or learn to pay more attention to elements in the beginning of the sequence then to elements at its end [15].

2.3.4 Training of artificial neural networks

Loss function

A supervised learning algorithm is a training set of $n$ training examples $x_{1:n} = x_1, x_2, \ldots, x_n$ together with corresponding labels $y_{1:n} = y_1, y_2, \ldots, y_n$. We here
assume that the desired inputs and outputs are vectors: \(x_{1:n}, y_{1:n}\). The goal of training a neural network is to return a function \(f()\) that accurately maps unseen input to their desired output, i.e., a function \(f()\) such that the predictions \(\hat{y} = f(x)\) are accurate. We introduce a loss function, quantifying the loss suffered when predicting \(\hat{y}\) while the true label is \(y\). A loss function \(L(\hat{y}, y)\) assigns a numerical score (a scalar) to a predicted output \(\hat{y}\) given the true expected output \(y\). Given a labeled training set \((x_{1:n}, y_{1:n})\), a per-instance loss function \(L\) and a parametrized function \(f(x; \Theta)\), where \(\Theta\) are the parameters of the neural network, we define the corpus-wide loss with respect to the parameters \(\Theta\) as the average loss over all training examples:

\[
L(\Theta) = \frac{1}{n} \sum_{i=1}^{n} L(f(x_i; \Theta), y_i).
\]

(2.17)

In this view, the training examples are fixed, and the values of the parameters determine the loss. The goal of the training algorithm is then to set the values of the parameters \(\Theta\) such that the value of \(L\) is minimized [15]:

\[
\hat{\Theta} = \arg\min_{\Theta} L(\Theta) = \arg\min_{\Theta} \frac{1}{n} \sum_{i=1}^{n} L(f(x_i; \Theta), y_i).
\]

(2.18)

An effective method for training linear models is using the SGD algorithm [25] or a variant of it.

**Regularization of parameters**

Equation 2.18 attempts to minimize loss at all costs. To prevent overfitting to the training data and making the algorithm more generalized by posing soft restrictions on the forms of the solution, which is done by using a function \(R(\Theta)\) taking as input the parameters and returning a scalar that reflect their "complexity" which we want to keep as low as possible. By adding \(R\) to the objective, the optimization problem becomes:

\[
\hat{\Theta} = \arg\min_{\Theta} \left( \frac{1}{n} \sum_{i=1}^{n} L(f(x_i; \Theta), y_i) + \rho R(\Theta) \right).
\]

(2.19)

The function \(R\) is called a *regularization term*. This poses restriction on the parameters, where large weights (emphasis on certain parts of the ANN structure) is punished by a larger loss. In this way, the model overfits less to the training data.

Another effective technique for preventing neural networks from overfitting the training data is dropout training [26][27]. The dropout method is designed to prevent the network from learning to rely on specific weights. It works by randomly dropping (setting to 0) a percentage of the neurons in the network (or in a specific layer) in each training batch.
Gradient-based optimization

Similar to linear models, neural networks are differentiable parameterized functions, and are trained using gradient-based optimization. The objective function for non-linear neural networks is not convex, and gradient-based methods may get stuck in a local minima. Still, gradient-based methods produce good results in practice.

In order to train the model, we need to solve the optimization problem in equation 2.18. An effective method for training linear models is using the Stochastic Gradient descent algorithm (SGD). It receives a function \( f() \) parametrized by \( \Theta \), a loss function \( L \), and desired input and output pairs \( x_{1:n}, y_{1:n} \). It then attempts to set the parameters \( \Theta \) such that the cumulative loss of \( f() \) on the training examples is small. Using mini batches of training samples, the error and gradients estimate is based on a sample size that can range between \( m = 1 \) and \( m = n \). This way the loss is attempted to be minimize with respect to several observations, instead of just one which makes the algorithm provide better estimates of the corpus-wide gradients.

Gradient calculation is central to the approach. The mathematics of gradient computation for neural networks are the same as those of linear models, simply following the chain-rule of differentiation. However, for complex networks this process can be laborious and error-prone. Fortunately, gradients can be efficiently and automatically computed using the back propagation algorithm [28] [29]. The back propagation algorithm is a fancy name for methodically computing the derivatives of a complex expression using the chain rule, while caching intermediary results [15].

2.3.5 Neural networks for regression

Consider a regression problem where the training data is on the form \((x_{1:n}, y_{1:n})\) where \(x_i\) represents the explanatory variables (word embeddings of textual data in the NLP case) and \(y_i\) the response variable. We also have \(x_{n+1}\) and want to predict the response \(y_{n+1}\). We put \( X = (x_1, y_1, \ldots, x_n, y_n, x_{n+1}) \) and \( \hat{Y} = \hat{y}_{n+1} \). The regression model is:

\[
\hat{Y}_i = f_\Theta(X),
\]

where \(f_\Theta\) is a neural network, parametrized by \( \Theta \). In order to maximize the probability of predicting the response \(\hat{Y} = \hat{y}_{n+1}\) the loss needs to be minimized.

Negative cosine proximity as loss function in regression

In NLP, word similarity is measured by cosine proximity as mentioned in section 2.2. Cosine proximity loss between an actual value \(Y\) and a predicted value \(\hat{Y}\) is defined as:

\[
\mathcal{L}(Y, \hat{Y}) = -\frac{Y \cdot \hat{Y}}{\|Y\|_2 \cdot \|\hat{Y}\|_2} = -\frac{\sum_{i=1}^{d_{out}} y(i) \cdot \hat{y}(i)}{\sqrt{\sum_{i=1}^{d_{out}} (y(i))^2} \cdot \sqrt{\sum_{i=1}^{d_{out}} (\hat{y}(i))^2}},
\]

(2.21)
where $Y = \{y^{(1)}, y^{(2)}, \ldots, y^{(d_{out})}\} \in \mathbb{R}^{d_{out}}$ and $\hat{Y} = \{\hat{y}^{(1)}, \hat{y}^{(2)}, \ldots, \hat{y}^{(d_{out})}\} \in \mathbb{R}^{d_{out}}$. It is the same as negative cosine similarity. Cosine similarity is a measure of similarity between two non-zero vectors of an inner product space that measures the cosine of the angle between them. Maximal "similarity" is achieved in the unit vector case when they are parallel and maximal dissimilarity is achieved when two vectors are orthogonal. Analogous to the cosine, the segments subtend a zero angle when maximally similar and the segments subtend a perpendicular angle when maximally dissimilar.
Chapter 3

Methods

This chapter is about the approach used in order to motivate the number of deep learning models chosen, and the architecture of these. By splitting the data into a training set and a validation set, the performance of the model architecture was made on the single label case, where only the observations that have one label were extracted from the data and trained on. In order to motivate the number of deep learning models to choose to model the multi label case, plots of the data and general industry standards were surveyed and used.

3.1 Data

General characteristics of the data

The data used in this thesis was provided by e-Avrop and contained approximately 30 000 public procurements from the e-Avrop website over the past years. Each data entry (observation) contained a title, description and corresponding CPV-code/codes. Figure 3.1 shows the 45 most general codes and the respective frequency of observations in each category, where all subcategories of every general code are contained in its respective parent code.
3.1. DATA

Figure 3.1: Frequency of observations in each of the most general categories. All observations of the 30,000 public procurements are plotted.

A clear imbalance of distribution in the training data is shown in figure 3.1, where the data points are far from uniformly distributed on the categories. In order to get a better view of the data at hand for the multi label case, the amount of labels (CPV-codes) per observation was plotted and is illustrated in Fig 3.2. Note that observations with zero CPV codes is a case that cannot be used in the training of a supervised model and was thus ignored in analysis and training.
3.2 Input space

In order to feed observations $x_{1:n} = x_1, x_2, \ldots, x_n$ to a neural network $f_{\Theta}(x_i) = \hat{y}_i$, the input sequence of words $x_i$ of a single observation is transformed via word embedding to be represented as numerical values. In order to get a sense of the appropriate input dimensions (input dimensions of an ANN must be fixed and predefined), the number of words per title/description are shown in Figure 3.3 and Figure 3.4.
3.2. INPUT SPACE

From figure 3.3 and Figure 3.4 we can see that \( \sim 20 \) words capture the title and \( \sim 400 \) words capture the description. Since a title is well-defined by a word count of up to \( \sim 20 \) words and a description is well-defined by a word count of up to \( \sim 400 \) words the input is to be a concatenated vector of 420 words. If the words of title + description is shorter than 420 words, the missing values are set to zero.
3.3 Output space

From Figure 3.1 we can draw the conclusion that the training data is not well distributed over the entire output space. One possible way of defining the labels of n observations is that the corresponding class label of the ith observation, \( y_i \), may be represented as a one-hot-vector:

\[
y_i = (y_1^i, \ldots, y^n_i),
\]

where all but one element are zero and \( y^C_i = 1 \) for the correct class. The objective is to construct a classifier that, given an observation \( x_i \), predicts the correct label. Since the training data in our is not well distributed over the entire output space, this approach is not considered, but instead a different approach is suggested. In section 2.3.5 the setup for neural networks in regression problems is presented, which is what is used as output of the ANN.

Transforming labels into word embeddings

For observations on the form \( (x_{1:n}, y_{1:n}) \) where \( x_i \) represents the explanatory variables (word embeddings of textual data in the NLP case) and \( y_i \) the response variable., regression via a neural network can be made if the labels \( y_{1:n} \) are transformed into word embedding vectors. Since the input \( x_i \), of the ith observation is originally transformed from a sequence of words into a sequence of numerical vector representations of words, the same can be done with the output \( y_i \). In Figure 3.5 the amount of words per CPV code description is plotted as a histogram.

![Histogram of words per CPV-code](image)

Figure 3.5: Histogram of number of codes per CPV code description.

For simplicity, the average of the words embeddings from the CPV code description are taken as the word embedding equivalent of the label. For example the label
3.4. LOSS FUNCTION

"03222321 - Apples" is transformed to the fastText word embedding of the word "apples". For CPV code descriptions consisting of several words such as "03222320 - Apples, pears and quinces", stop words such as "and", "him", "she" etc. are removed before calculating the word embedding average. The word embedding for "03222320 - Apples, pears and quinces" therefore becomes the average:

\[
\text{embedding}(03222320) = \frac{\text{embedding}(\text{Apples}) + \text{embedding}(\text{Pears}) + \text{embedding}(\text{Quinces})}{3}.
\]

The output dimension is then fixed and is of dimension 300, the dimension of the fastText word embeddings.

3.4 Loss function

In order to maximize the probability of predicting the response \( \hat{Y} = \hat{y}_{n+1} \), the loss needs to be minimized during training. Since word similarity in the framework of word embeddings is measured by cosine similarity, a natural loss function to use is the negative cosine proximity, defined in equation 2.21. We have now defined that for the actual value \( Y \) and a predicted value \( \hat{Y} \) the dimensions are \( d_{out} = 300 \). Large deviations from the cosine similarity will have large values of the loss function, which is congruent with the fastText word embedding framework.

The reason the loss function is chosen to be cosine proximity is because of the nature of the problem. When the observation is categorized with a specific code, we want a measure which defines the proximity to the specific code even if the predicted value \( \hat{Y} \) is not exactly the same as the true value \( Y \). The CPV-code of a training observation is not always "correct" in the sense that a better code can be appropriate than what the training data suggests. In this case, the cosine proximity provides a measure of how close the regression is in the vector space of the words embeddings.

3.5 Validation

The validation of the models was done using the validation set approach. The data (observations and labels) were randomly shuffled with a seed in order to preserve the same training/validation split across all experiments. The split was made 80/20 according to the general guidelines often used in similar problems [30].
3.6 Training models

3.6.1 Single label case

In sections 3.2 and 3.3 we defined the fixed input and output dimensions of the neural network. In table 3.1 the final input and output dimensions are displayed.

<table>
<thead>
<tr>
<th>Matrix/concatenated</th>
<th>Input layer</th>
<th>Output layer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrix</td>
<td>420x300</td>
<td>300</td>
</tr>
<tr>
<td>Concatenated</td>
<td>126 000</td>
<td>300</td>
</tr>
</tbody>
</table>

Table 3.1: Table of models tested in the single-label case.

In order to get a sense of what model architecture fits the problem the best, and since the most frequent amount of labels for each observation is a single label (see Figure 3.2) it is a good starting point of comparing different model architectures. With a total of 16 470 observations being single label, the entire data set for evaluating the performance of deep learning models was split with 80% training set and 20% validation set. The models evaluated are presented in table 3.2.
3.6. TRAINING MODELS

<table>
<thead>
<tr>
<th>Model name</th>
<th>GRU max</th>
<th>LSTM max</th>
<th>LSTM conv</th>
<th>GRU att</th>
<th>LSTM att</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>420x300</td>
<td>420x300</td>
<td>420x300</td>
<td>420x300</td>
<td>420x300</td>
</tr>
<tr>
<td>Layer 1</td>
<td>GRU</td>
<td>LSTM</td>
<td>GRU/LSTM</td>
<td>GRU</td>
<td>LSTM</td>
</tr>
<tr>
<td>Layer 2</td>
<td>Max</td>
<td>Max</td>
<td>Max/Avg</td>
<td>Timedistr</td>
<td>Timedistr</td>
</tr>
<tr>
<td>Layer 3</td>
<td>Dense(1024)</td>
<td>Dense(1024)</td>
<td>Max/Avg</td>
<td>Attention</td>
<td>Attention</td>
</tr>
<tr>
<td>Layer 4</td>
<td>Dense(300)</td>
<td>Dense(300)</td>
<td>Max/Avg</td>
<td>Dense(300)</td>
<td>Dense(300)</td>
</tr>
<tr>
<td>Layer 5</td>
<td>-</td>
<td>-</td>
<td>Dense(512)</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Layer 6</td>
<td>-</td>
<td>-</td>
<td>Dense(300)</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Layer 7</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 3.2: Table of input/output dimensions of the ANN.

Note all the recurrent layers are bidirectional. "Timedistr" layer in Table 3.2 is a dense layer of dimension 420x1024. For the last Dense layer in all the models, the tanh is used as activation function because of word embeddings taking values between -1 and 1 (analogous to the tanh activation function).

### 3.6.2 Multi label case

In the multi label case, the goal is to create classifiers with "intent" which is to create classifiers which tend to focus on different behaviour. In the case of public procurements and the CPV-code categorization, the codes are structured according to depth in the CPV tree, general codes are higher up in the tree cover more categories, while more detailed categories are found lower in the tree. Each node has a level, which corresponds to the depth of the node.

![Figure 3.7: Depth of nodes illustrated.](image)

We thus create several deep learning models, with datapoints that are on average, at different levels in the tree. By dividing a multi label observation into several observations, each with a single corresponding observation - label pair. These pairs
are constructed in a way such that observation - label pairs which share similar characteristics of chosen type are grouped together. By consulting with e-Avrop and their employees that manually categorize public procurements, a good rule of thumb is to code a public procurement with 2-4 CPV codes. By looking at the data from figure 3.2, it is clear that most observations fall within this range. A cut off was therefore chosen at observations that have more than six labels, in order to filter observations of low quality. The deep learning models with intent were made with one general model (red zone in Figure 3.7) and one granular model (blue zone in 3.7). In order to compare with deep learning models with "no intent", observations that fall in the middle were divided into two data sets, used as training data for two models that handle ties. The models are described as:

1. **General Model**: Takes the most general codes in the multi label case. Out of several labels on a single observation, this deep learning model is trained on the label with the most general CPV-code.

2. **Detailed Model**: Takes the most detailed codes in the multi label case. Out of several labels on a single observation, this deep learning model is trained on the label with the most detailed CPV-code.

3. **Ties.1 Model**: If there are labels that have the same level of detail (which cannot be used for training in the general/detailed models), this model uses the observation as training data. If there exists only one such label, for example if we have 3 labels which are: general, detailed and the last label is in between, this model has a 50% chance of using it in its training set. Otherwise, "Ties.2 model" uses it in its training set.

4. **Ties.2 Model**: Same as for "Ties.1" model, receives the labels that are not considered general or detailed with a 50% probability with model "Ties.1".

The algorithm works as follows (X are observations, Y are labels):
3.6. TRAINING MODELS

Algorithm 1: How the multi label training sets are created

\[
\begin{align*}
\text{set } & (X_{\text{general}}) = \text{emptylist} \\
\text{set } & (Y_{\text{general}}) = \text{emptylist} \\
\text{set } & (X_{\text{detailed}}) = \text{emptylist} \\
\text{set } & (Y_{\text{detailed}}) = \text{emptylist} \\
\text{set } & (X_{\text{Ties,1}}) = \text{emptylist} \\
\text{set } & (Y_{\text{Ties,1}}) = \text{emptylist} \\
\text{set } & (X_{\text{Ties,2}}) = \text{emptylist} \\
\text{set } & (Y_{\text{Ties,2}}) = \text{emptylist} \\
\text{for Each multi label observation } X_i \text{ do} \\
& \text{instructions;} \\
& \quad \text{if } \#\text{labels} < 7 \text{ then} \\
& \quad \quad \text{if labels have different level then} \\
& \quad \quad \quad \text{append } X_i \text{ to } (X_{\text{general}}); \\
& \quad \quad \quad \text{append } Y_{\text{general}} \text{ to } (Y_{\text{general}}); \\
& \quad \quad \quad \text{append } X_i \text{ to } (X_{\text{detailed}}); \\
& \quad \quad \quad \text{append } Y_{\text{detailed}} \text{ to } (Y_{\text{detailed}}); \\
& \quad \quad \quad \text{what is left of the labels append to } X_{\text{Ties,1}}, Y_{\text{Ties,1}} \text{ or } X_{\text{Ties,2}}, Y_{\text{Ties,2}} \text{ at random;} \\
& \quad \quad \text{else} \\
& \quad \quad \quad \text{append } X_{\text{Ties,1}}, Y_{\text{Ties,1}} \text{ or } X_{\text{Ties,2}}, Y_{\text{Ties,2}} \text{ at random;} \\
& \quad \text{end} \\
& \text{else} \\
& \quad \text{skip;} \\
& \text{end} \\
\end{align*}
\]
Chapter 4

Results

Evaluating the deep learning models to be adapted to the NLP problem at hand is crucial in order to evaluate the multi label case. In this chapter we first of all go through the performance of the models presented in table 3.2. After this, the multi label performance and results are presented with the deep learning model chosen from the single label case.

4.1 Training of deep learning models in the single labels case

Performance is evaluated with respect to the loss function of the validation set.

![Training/Validation Loss for GRU_max model](image)

Figure 4.1: Training and validation loss of the GRU_max model.
4.1. TRAINING OF DEEP LEARNING MODELS IN THE SINGLE LABELS CASE

Figure 4.2: Training and validation loss of the LSTM_max model.

Figure 4.3: Training and validation loss of the LSTM_conv model.
CHAPTER 4. RESULTS

Figure 4.4: Training and validation loss of the GRU\textsubscript{att} model.

Figure 4.5: Training and validation loss of the LSTM\textsubscript{att} model.

From Figures 4.1 - 4.5 the lowest loss is obtained by the GRU\textsubscript{att} model with a validation loss just below \textminus0.7. This model is therefore used in the multi label case for analysis.
4.2. PERFORMANCE OF THE DEEP LEARNING MODELS IN USING THE MULTI LABEL DATA SET

4.2 Performance of the deep learning models in using the multi label data set

Four different data sets for different deep learning models

The models presented in section 3.6.2 each have a unique data set for training/validation which were generated according to algorithm 1, but with the same architecture which is that of the GRU_att model presented in 3.2. The performance of the models are presented in the figures below.

![Training loss on dataset for general model](image)

Figure 4.6: Training and validation loss of the GRU_att model for the "general" data set.
Figure 4.7: Training and validation loss of the GRU\_att model for the "Ties\_1" data set.

Figure 4.8: Training and validation loss of the GRU\_att model for the "Ties\_2" data set.
4.2. PERFORMANCE OF THE DEEP LEARNING MODELS IN USING THE MULTI LABEL DATA SET

From the plots of losses in Figures 4.6 - 4.9 we can see a clear reduction in performance compared to the single label case in Figures 4.1 - 4.5. This is because there is not enough data in the multi label case to train the classifiers correctly. In order to provide more information to the deep learning models, every data set (general, Ties_1, Ties_2 and detailed) received the entire single label data set on top of their respective data sets. The summed data sets were again shuffled and split into 80/20 training/validation sets.

Figure 4.9: Training and validation loss of the GRU_att model for the “detailed” data set.

**Combining each individual data set with the single label data set**

From the plots of losses in Figures 4.6 - 4.9 we can see a clear reduction in performance compared to the single label case in Figures 4.1 - 4.5. This is because there is not enough data in the multi label case to train the classifiers correctly. In order to provide more information to the deep learning models, every data set (general, Ties_1, Ties_2 and detailed) received the entire single label data set on top of their respective data sets. The summed data sets were again shuffled and split into 80/20 training/validation sets.
Figure 4.10: Training and validation loss of the GRU_att model for the "general" data set.

Figure 4.11: Training and validation loss of the GRU_att model for the "Ties_1" data set.
4.2. PERFORMANCE OF THE DEEP LEARNING MODELS IN USING THE MULTI LABEL DATA SET

Figure 4.12: Training and validation loss of the GRU \texttt{att} model for the "Ties\_2" data set.

Figure 4.13: Training and validation loss of the GRU \texttt{att} model for the "detailed" data set.

From Figures 4.10 - 4.13 there is now a clear overall increase in performance compared to the case where only the model specific data sets were used. The plots only show loss plotted against epoch numbers, but not whether the deep learning models actually behave differently when classifying data. Next, the behaviour of classification is investigated.
Verifying the intent assumption of the deep learning models

In order to investigate the behaviour of the deep learning models, the single label data set added to each unique multi label data set was used. After classifying and observation, the depth of the label that the classifier chose was recorded. The combined data sets were used to train each model and the depth of the classifications are shown in the histograms below.

![Histogram of general model](image1)
![Histogram of detailed model](image2)
![Histogram of Ties_1 model](image3)
![Histogram of Ties_2 model](image4)

Figure 4.14: Histogram of classification depth for the 4 different deep learning models.

The Figures 4.14a - 4.14d indeed show a clear difference in behaviour (especially in the case of general to detailed model). This difference in intent is also depicted in Table 4.1.
4.2. PERFORMANCE OF THE DEEP LEARNING MODELS IN USING THE MULTI
LABEL DATA SET

<table>
<thead>
<tr>
<th>Data set</th>
<th>Average classification level</th>
<th>Average depth in % of total tree depth</th>
</tr>
</thead>
<tbody>
<tr>
<td>General</td>
<td>2.536</td>
<td>36.2%</td>
</tr>
<tr>
<td>Ties_1</td>
<td>2.891</td>
<td>41.3%</td>
</tr>
<tr>
<td>Ties_2</td>
<td>2.821</td>
<td>40.3%</td>
</tr>
<tr>
<td>Detailed</td>
<td>3.132</td>
<td>44.7%</td>
</tr>
</tbody>
</table>

Table 4.1: Table of difference in average classification depth between the different deep learning classifiers.
Chapter 5

Discussion

5.1 Performance of neural networks

The set of deep learning models presented in Table 3.2 were shown to all have similar performance on the task of single label classification. Several other models were tested, with different structures and/or hyperparameters. These models were left out because of the obvious decrease in performance compared to the models used in this thesis.

The applicability of the deep learning models on the particular data set was therefore both evaluated before the models presented in Table 3.2 were chosen, and carefully evaluated among the models presented. The assumption where the encoder-decoder models performed best (LSTM encoder) was verified with the plots in section 4.1. The effectiveness and applicability of the classifiers was thus established in the single label case so that different classifiers could be evaluated, in order to later on create the set of classifiers which would be a base for the multi label case. The models trained were based on the architectures that were researched in several papers which explored deep learning for text classification, papers which have contributed to setting the standard of text classification in modern NLP problems [32][33][34][35].

5.1.1 Input space

In image classification, pre-training CNN’s on large data sets of images have been shown to provide “general” knowledge in image classification tasks that can be transferred and applied to a particular problem in the same area of domain [36]. In many cases, these synergies can be exploited by architectures that rely on similar components. Recently in the area of NLP, significant gains have been made through transfer and multi-task learning between synergistic tasks. Contextualized word/sentence embeddings is an example of transferred learning which has proven a general increase in text classification accuracy.

The word embeddings used as an input to the neural networks are, as mentioned earlier, n-gram representations of words in a real number numerical vector space.
However, the context in which the words are written does not directly change the vector representation of the words. In the experiment done in this thesis, the context is rather picked up by the biRNN encoder. In order for this context to be inherently existent in the word embedding, more complex structures need to be used. The details of contextualized word/sentence embeddings are not explained further in this thesis, but are worth mentioning as a possible improvement of the results [37][38].

5.1.2 Output space

The output space chosen in this thesis was, as explained in section 3.3, the average word embedding of the CPV-code description. This is a very simplified version in order to make it possible for an ANN regression to target any label in the label space. One issue of this is when the label description does not transfer well to the word embedding vector space. For example the CPV-code 09131000 - "Aviation kerosene" ("Flygfotogen" in swedish) becomes most similar to the word "Aviation photography" ("Flygfotografi" in swedish) which is the wrong interpretation. Out of all the 9 453 CPV codes, many are bound to be incorrect which means that the classifier would learn the connection between observation-label incorrectly in many cases.

5.2 Deep learning classifiers with intent

In the multi label case, the observations with several labels were transformed into single label observations, all with labels of different characteristics (levels of depth in the CPV-tree). In the case where only the labels from the multi label data set were used (section 4.2), the classifiers showed a significant reduction in performance compared to the case of single label observations in section 4.1. Because the smaller data set for each classifier generally was not enough to properly train each classifier, the single label data set was used as well, combining it with each individual data set.

Interestingly, even when the single label data set was added, the classifiers still showed a difference in behaviour when classifying the texts. With the difference in intent in classification depth shown in table 4.1, there is a clear distinction between the most general and most granular average classification depth. If the data set at hand was to be more spread out among the 9 453 CPV-codes (or in any other text classification problem with a vast number of labels), a possibility would be to create several classifiers which handle different regions in the label space. These classifiers could be divided by depth and width, each handling a different region in the label space.

Another point worth mentioning is that in the case of the four different deep learning models with intent, the most general model performed the best. Why exactly this is the case is difficult to know. According to the employees which today manually categorize each text, the optimal number of CPV-codes for a public
procurement is 2-4. When a text is categorized with only one label, that label is often general in order to capture most of the characteristics with only that specific label. A theory of why the general data set performs best is therefore that the most general label of the multi label case is just an extension of the single label observations, providing a more extensive training set.

5.3 Future work

In terms of the input to the deep learning models, the only word embedding tested in the experiments was the fastText word embedding. Initially, the word2vec model was used but since the CPV-code descriptions contained many out-of-vocabulary words, the n-gram model with fastText was used. Other word embeddings might achieve different results and with the recent achievements in transfer learning within NLP, contextual embeddings such as BERT (Bidirectional Encoder Representations from Transformers) might improve the classification task at hand.

In order to make the model more robust to unseen data, data augmentation might be of use in order to compensate for the lack of training data in some areas of the CPV-tree.

All of this is perhaps futile work if the labels (as discussed in section 5.1.2) often have the wrong vector representation. This could be handled in a number of ways, with perhaps the most rewarding way to manually check through the labels. Another way could be to incorporate the word embedding with context model on the labels at hand.

The classifier with intent approach could also be handled in different ways. One could concatenate several deep learning models. This would filter the first level of the CPV-tree with a deep learning model in order to give some general category suggestion for the regression models. After that, different regions could use a classifier with intent, classifiers could also be used laterally instead of vertically, to handle different characteristics of the data. These suggestions might give better performance and with a larger and more spread out data set, the number of classifiers with intent could be increased. Thus, a data set containing training data which would be more evenly spread among the labels might incorporate both lateral and vertical classifiers with intent.
Chapter 6

Conclusion

The approach of transforming labels into vectors and using a deep learning model for vector regression worked quite well on the data set. When practically used on data which had yet not been seen (public procurements from the website), the models were often more accurate within categories where there was a lot of data available (construction work for example). This performance evaluation is of course subjective, but with the understanding of natural language that humans have, the performance evaluation at some point needs to be subjective, when a machine is taking over a previously manual task. In the future, there is definitely room for a potential automatic public procurements categorization model, aiding manual categorization in order to make quicker and more efficient text classification possible.

The classifiers that used different data sets which were generated according to algorithm 1, indeed had different average depth of classification. This was in line with what was logically expected of the different data sets at hand. A deep learning regression model with a vast output space such as the one used in this thesis, inherently incorporates a complexity which makes it difficult to predict the behaviour of these classifiers beforehand. The fact that there is such a distinct difference between the general and more detailed model, suggests that the approach is justified and that it indeed is a legitimate approach for this text classification problem. This approach can perhaps also be applied to other instances of multi label classification where different classifiers have different intent, each capturing parts of the labeling problem at hand.
Bibliography


